Devillers Tom
Paternotte Mattéo

Dossier technique LoRaWAN

Introduction :

Ce projet portera sur la technologie LoRaWAN, un protocole de communication sans fil longue portée et basse consommation, particulièrement adapté aux applications IoT. Nous procéderons à l'acquisition de données notamment d'un capteur de température, un potentiomètre, un slider et une balance pour mesurer la masse. Ces données seront affichées à un écran et transmises via le réseau LoRa à un serveur LoRaWAN privé, hébergé sur la plateforme The Things Network (TTN).

Librairies utilisées :

Arduino.h : Fonctions de base Arduino.
Adafruit_GFX & ST7735 : Gestion de l'affichage graphique sur écran TFT.
SPI : Communication SPI pour le transfert de données.
HX711 : pour la balance.
Adafruit_SleepyDog : Gestion du Watchdog .
MKRWAN_v2 : Communication LoRaWAN.
RTCZero : Gestion de l'horloge en temps réel / veille profonde.
Adafruit_MCP9808 : Capteur de température.

Explication du code sur la carte :

Nos fonction principales :

setup() :

- Initialise les composants : écran, balance, capteur de température, watchdog .
- Calibre la balance pour mesurer précisément les poids.
- Configure l'écran pour afficher les données.
- Configure LoRaWAN pour envoyer des messages en utilisant la méthode sécurisée OTAA (Over-The-Air Activation).
- Effectuer des test (écran et LoRaWAN)

balance() :

- Récupère la valeur de la masse en grammes.
- Convertit la masse en valeur entière (pour simplifier l'affichage et la transmission).

testAnalog() :

- Récupère les données du potentiomètre, du slider, de la température et de la balance.
- Efface et met à jour une zone de l'écran pour afficher les nouvelles mesures.
- Crée une trame compacte contenant ces valeurs (masse; potentiomètre; slider; température).

testdrawtext() :

- Affiche du texte sur l'écran avec des paramètres de couleur et de position.

main() :

- Réinitialise le watchdog et met l'appareil en veille prolongée pendant 13 minutes et 33 secondes. Comme l'émission dure 8 secondes et représente 1 % du temps total, la période doit être de T=Th/a=8/0,01=800s = 13m et 33 secondes.
- Lit les capteurs (température, potentiomètre, slider, balance).
- Affiche les données sur l'écran TFT.
- Envoie la trame LoRa contenant les mesures (Valeur_masse; Valeur_pot; Valeur_slider; Valeur_temp).

  Cycle d'exécution :

  - Lecture des capteurs (potentiomètre, slider, température, masse).
  - Mise à jour de l'écran avec les nouvelles données.
  - Création et envoi de la trame LoRa contenant toutes les mesures.
  - Veille profonde pendant 13 minutes et 33 secondes.

Explication du payload formatter :

Un payload formatter est une fonction utilisée pour décoder et formater les données envoyées par la carte. Il permet de transformer ces données hexadécimal en une information lisible et exploitable.

Code :

```
function Decoder(bytes, port) { //fonction qui convertit un buffer en ascii sans b64
  const longeur_buff=bytes.length//mesurer la taille du buffer
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Strin
g/length
  let a=0 //variable a en int pour ajouter au string dans la boucle
  //pour convertire :
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/fr
omCharCode
```

```
  let ascii= "" //variable ascii en string pour rassembler
  while (a<longeur_buff){// tant l'index a est inférieur à la longueur du buffer alors
faire
  ascii=ascii+String.fromCharCode(bytes[a]); // rassemble chaque caractère pour en faire
une chaîne
  a++;// augmenter l'index
  }
  // sépare la chaîne ascii par ";"
  let splitAscii = ascii.split(";");

  // recup les éléments
  let masse_v = splitAscii[0]+ " G";
  let pot_v = splitAscii[1];
  let slider_v = splitAscii[2];
  let temp_v = splitAscii[3]+ " °C";
  return {
    Masse : masse_v,
    pot : pot_v,
    slider : slider_v,
    temp : temp_v
    //,buffer : longeur_buff
  };
}
```

Explication du code du payload formatter :

Les variables utilisées

- longeur_buff : Cette variable contient la taille du buffer bytes, c'est-à-dire le nombre d'éléments qu'il renferme. Elle permet de déterminer la quantité de données à traiter.

- a : Un compteur initialisé à 0, utilisé dans une boucle pour parcourir chaque élément du buffer.

- ascii : Une chaîne de caractères initialement vide, qui servira à assembler les caractères convertis du format hexadécimal vers ASCII (texte).

Conversion des données du buffer en chaîne de caractères :

La transformation des données du buffer hexadécimal en texte ASCII se fait à l'aide d'une boucle while. Cette boucle s'exécute tant que l'index **a** reste inférieur à la **taille du buffer**.

- Conversion des valeurs :

  À chaque passage, la valeur du buffer **bytes[a]** (valeur d'un octet en hexa correspondant à l'index **a**) est convertie en un caractère **ASCII** grâce à la fonction **String.fromCharCode(bytes[a])**. Cette fonction traduit un code hexa en son équivalent en caractères ASCII.Par exemple, si bytes[a] vaut 0x31, la conversion donnera le caractère '1'.

- Construction de la chaîne ASCII :

  Chaque caractère obtenu est ajouté à la variable ascii. Ainsi, à chaque passage, la chaîne se construit progressivement avec l'opération suivante : **ascii = ascii + String.fromCharCode(bytes[a]).**

  À la fin de cette boucle, la variable ascii contient une chaîne complète,
  par exemple :**"100;50;30;22,50"**, représentant les données transmises par la carte.

Séparation des données :

Une fois la chaîne ASCII formée, la méthode **split(";")** est utilisée pour découper cette chaîne en plusieurs parties, en prenant le caractère ';' comme séparateur. Chaque partie correspond à une mesure.

- Exemple :

  Pour la trame reçue **"100;50;30;22,50"**, les données se traduisent comme cela :

  100 : La masse en grammes.
  50 : La valeur du potentiomètre.
  30 : La position du slider.
  22,50 : La température en degrés Celsius.

Pour mieux comprendre nous allons ajouter des caractères à la masse et à la température :

La masse est annotée avec " G" pour indiquer qu'elle est exprimée en grammes.
La température est complétée par " °C" pour préciser qu'elle correspond à des degrés Celsius.

Résultat :

- Enfin, la fonction retourne un objet Payload contenant toutes les mesures :

```
{
  Masse: "100 G",
  Pot: "50",
  Slider: "30",
  Temp: "22,50 °C"
}
```

Test du projet :

10h36:21s : 303B303B3136333B31392E3139
10h34:47s : 3836373B303B303B31392E3132
10h33:11s : 303B39393B3131393B31392E3036
10h33:04s : 4142434461626364

Nous avons réduit la durée du mode de veille profond pour obtenir le plus de trames possible, et l'avons fixée à 1 minute et 33 secondes. Tout d'abord, nous effectuons le test avec le caractère "ABCDabcd", puis ensuite nous envoyons nos valeurs de masse, de potentiomètre, de slider et de température toutes les 1 minute et 33 secondes.

Vérifions que la mise en veille et conforme au 1 minute et 33 seconde :

Convertir chaque temps en secondes :

- 10h36:21s = 10*3600 + 36*60 + 21 = 38181 secondes
- 10h34:47s = 10×3600 + 34×60 + 47 = 38087 secondes.
- 10h33:11s = 10×3600 + 33×60 + 11 =37991 secondes.
- 10h33:04s =10×3600 +33×60 + 4 =37984 secondes.

Calcul entre chaque intervalle de temps :

- Entre 10h36:21 et 10h34:47 :
  - 38181 − 38087 = 94 secondes(1 minute 34 secondes).
- Entre 10h34:47 et 10h33:11 :
  - 38087 − 37991 = 96 secondes(1 minute 36 secondes).
- Entre 10h33:11 et 10h33:04 :
  - 37991 − 37984 = 7 secondes.

Pour conclure, le temps est bien respecté. Nous avons un écart de 1 à 2s qui peut s'expliquer par le temps de transmission et le temps de décodage.

Conclusion :

Ce projet nous a permis de découvrir et de mieux comprendre plusieurs technologies et concepts. Nous avons appris à utiliser la communication LoRaWAN, à gérer le mode watchdog sleep, et à intégrer le concept de duty cycle dans nos programmes. Cela nous a aussi donné l'occasion de nous familiariser davantage avec le langage JavaScript avec le payload formatter.

En travaillant sur ce projet, nous avons mieux compris comment fonctionnent les transmissions LoRa et comment un serveur LoRaWAN reçoit et traite les données. Cela nous a également permis de renforcer nos compétences en programmation.